УДК 519.4

Е.М. Насіров
Київський національний університет імені Тараса Шевченка, Україна
Україна, 03680, м. Київ, пр-т. Глушкова 4д

МОДЕЛЬ ПАРАЛЕЛІЗАЦІЇ НЕВІД'ЄМНОЇ ФАКТОРИЗАЦІЇ РОЗРІДЖЕНИХ МАТРИЦЬ НАДВЕЛИКОЇ РОЗМІРНОСТІ

E.M. Nasirov
Kyiv National Taras Shevchenko University, Ukraine
Ukraine, 03680, c. Kyiv, Glushkova 4d

MODEL FOR PARALLEL NON-NEGATIVE SPARSE EXTRA-LARGE MATRIX FACTORIZATION

Э.М. Насиров
Киевский национальный университет имени Тараса Шевченко, Украина
Украина, 03680, г. Киев, пр-т. Глушкова 4д

МОДЕЛЬ ПАРАЛЛЕЛИЗАЦИИ НЕОТРИЦАТЕЛЬНОЙ ФАКТОРИЗАЦИИ РАЗРЯЖЕННЫХ МАТРИЦ СВЕРХБОЛЬШОЙ РАЗМЕРНОСТИ

У роботі описано побудову моделі паралелізації обчислення невід’ємної факторизації розріджених матриць надвеликої розмірності. Реалізації запропонованих моделей були порівняні в обробці надвеликої матриці.
Ключові слова: лінгвістика, паралелізація, невід’ємна факторизація матриць, GPU.

This paper proposes parallel methods of non-negative huge sparse matrix factorization. The implementation of proposed methods was tested and compared on huge matrix
Key words: computational linguistics, parallel computations, non-negative matrix factorization.

В работе описано построение модели параллелизация вычисления неотрицательной факторизации разреженных матриц сверхбольшой размерности. Реализации предложенных моделей были сравнены в обработке сверхбольшой матрицы.
Ключевые слова: лингвистика, параллелизация, неотрицательная факторизация матриц, GPU.

Сьогодні невід’ємна факторизація матриць і тензорів є дуже популярною технологією в штучному інтелекті взагалі, та, зокрема, в комп’ютерній лінгвістиці. Використовуючи невід’ємну факторизацію в рамках парадигми латентно-семантичного аналізу, комп’ютерні лінгвісти застосовують даний підхід для розв’язання таких прикладних задач, як класифікація, кластеризація текстів та термінів[1, 2], побудова мір семантичної близькості[3], автоматичне виділення лінгвістичних структур та відношень (Selectional Preferences [4] and Verb Sub-Categorization Frames [5]) та багато інших.

Дана робота описує побудову моделі паралелізації обчислення невід’ємної факторизації розріджених матриць надвеликої розмірності, що представляє особливий інтерес та актуальність для її застосування у великих NLP системах загального тематичного напрямку, необмежених використанням лише для вузьких предметних областей.

Задача невід’ємної факторизації розріджених матриць надвеликої розмірності постала в процесі розробки системи визначення міри семантичної близькості-зв’язності за технологією латентного семантичного аналізу[6]. Для побудови системи був оброблений великий текстовий корпус статей англомовної

Вікіпедії. Обробка корпусу полягала у лексичному аналізі та лематизації лексем речень статей та в обчисленні частот вживання множини слів та словосполучень англійської мови в складі різних статей англомовної Вікіпедії. У результаті була побудована велика матриця Слова×Статті, яка містила частотну оцінку вживання слів в текстах статей Вікіпедії. Розмірність матриці дорівнює 2,437,234 слів-словосполучень на 4,475,180 статей англомовної Вікіпедії. Після цього для усунення з матриці випадкових даних був встановлений пороговий рівень T=3, частотні оцінки в матриці, які менші за пороговий рівень T, були обнулені). У результаті була побудована розріджена матриця надвеликого розміру – 156,236,043 ненульових елементів при розмірі 2,437,234×4,475,180. Для невід’ємної факторизації розрідженої матриці такого розміру знадобилася розробка спеціальної моделі паралелізації матричних обчислень, яка була реалізована із застосуванням паралельних розподілених обчислень та обчислень на GPU. Низка реалізацій факторизації матриць була вже реалізована і представлена у [7,8,9,13]. Але жодна з розроблених моделей не є допустимою для поставленої задачі. Деякі з них [7,8,9] не задовольняють потреби розмірності матриці. Модель, запропонована в [13], проводить факторизацію матриць запропонованого розміру за задовільний час, але потребує занадто великих розрахункових комп’ютерних потужностей, що є не завжди доступним.

Алгоритм невід’ємної матричної факторизації

Алгоритм невід’ємної матричної факторизації[11] виконує декомпозицію невід’ємної матриці V на невід’ємні матриці W та H таким чином, щоб:

$$V \approx WH$$

Як функція оцінки μ може бути використана функція виміру відстані між двома невід’ємними матрицями. Однією з таких мір є квадрат Евклідової метрики:

$$\mu = \|A - B\|^2 = \sum_{ij} (A_{ij} - B_{ij})^2$$

Така цільова функція обмежена знизу. Нижня границя 0 досягається тоді і тільки тоді коли

$$A = B$$

Отже, при використанні Евклідової метрики, факторизація матриці полягає в мінімізації $\|V - WH\|^2$ за умови невід’ємності W та H.

Така цільова функція не зростаюча за наступних правил:

$$H_{ij} \leftarrow H_{ij} \frac{(W^T V)_{ij}}{(W^T W H)_{ij}} \quad (1)$$

$$W_{ij} \leftarrow W_{ij} \frac{(V H^T)_{ij}}{(W H H^T)_{ij}} \quad (2)$$

Виконання ітерацій алгоритму продовжується до тих пір, поки не буде досягнута стаціонарна точка, або не буде виконана максимальна кількість ітерацій.

Аналіз моделі

У таблиці 1 представлено необхідні об’єми пам’яті для зберігання матриць W та H при різних k при використанні типу даних float (32bit).

Описаний алгоритм на кожній ітерації потребує у 2 рази більше пам’яті для збережень матриць (не враховуючи потреби на збереження початкової матриці V). Враховуючи такі потреби в пам'яті, для виконання алгоритму на одному комп'ютері необхідно проводити збереження даних на жорсткий диск. Далі буде

розглянуто та порівняно 2 підходи до паралелізації алгоритму: локальний та розподілений.

Таблиця 1. Необхідні об'єми пам'яті для зберігання матриць W та H при різних k

K	*100*	*200*	*300*
W	0.98Gb	1.95Gb	2.92Gb
H	1.79Gb	3.58Gb	5.37Gb
Всього	2.76Gb	5.53Gb	8.29Gb

Паралелізація алгоритму з використанням GPU

Для спрощення, зробимо заміну в (1) та (2): $H^{'} = H^{T}$. Отримаємо

$$(H'_t)_{ij} = (H'_{t-1})_{ij}\frac{\left(V^T W_{t-1}\right)_{ij}}{\left(H'_{t-1} W_{t-1}^T W_{t-1}\right)_{ij}} \quad (4)$$

$$(W_t)_{ij} = (W_{t-1})_{ij}\frac{\left(VH_t^{'}\right)_{ij}}{\left(W_{t-1} H_t^{'T} H_t^{'}\right)_{ij}} \quad (5)$$

Таким чином, обидві формули (4) та (5) ми можемо представити в одному вигляді, завдяки заміні H', W та V^T, або W, H' та V на A, B та S відповідно:

$$A_{ij} = A_{ij}\frac{(SB)_{ij}}{\left(AB^T B\right)_{ij}} \quad (6)$$

Формула (6) може бути представлена як послідовність чотирьох кроків (7):

$$C = SB \quad (7a)$$

$$K = B^T B \quad (7b)$$

$$D = AK \quad (7c)$$

$$A_{ij} = Aij\frac{C_{ij}}{D_{ij}} \quad (7d)$$

Такий порядок обчислень є оптимальним для виразу (6). Ці кроки мають обчислювальну складність $O(k * (nnz(S) + n))$, $O(k^2 m)$, $O(k^2 n)$ та $O(kn)$ відповідно, де $nnz(S)$ – кількість ненульових елементів матриці S. Перші три кроки(7a-7c) підтримуються бібліотеками CUDA cuSPARSE[17] та cuBLAS[16]. Четвертий крок (7d) для виконання потребує реалізації власного ядра GPU, але в той же час, це відносно швидка операція і тому може бути виконана на CPU.

Так як матриці занадто великі для збереження в пам'яті GPU, ці операції(7a-7d) повинні виконуватись над частинами, з врахуванням необхідності зменшення об'ємів обміну даними з графічним адаптером.

Для виразу (7a) повинні розкласти матриці $S = \left(S_1^{'}\middle|S_2^{'}\middle|\dots\middle|S_t^{'}\right)^T$ та $B = (B_1|B_2|\dots|B_r)$ та підрахувати C:

$$C = \begin{pmatrix} S_1^{'}B_1 & \cdots & S_1^{'}B_r \\ \vdots & \ddots & \vdots \\ S_t^{'}B_1 & \cdots & S_t^{'}B_r \end{pmatrix} \quad (8)$$

Для виразу (7b) варто розглядати $B = (B'_1|B'_2|\dots|B'_t)$, а отже $K = B'^T_1 B^{'}_1 \dots B`^T_t B`_t$. Підрахунок цього виразу не потребує додаткового завантаження будь-яких матриць у пам'ять графічного адаптера після виконання (7a).

Для підрахунку (7c) ми можемо залишити в пам'яті GPU матрицю K і помножити матрицю A як стовпчик блоків.

Також можна зменшити використання пам'яті завдяки комбінуванню (7d) та (7c), так як (7d) – поелементне правило і єдине інше правило, в якому використовується матриця A – (7c). Не буде потреби зберігати в пам'яті матрицю D, якщо виконувати (7d) на блоці з матриці A, який необхідний для обчислення певного блоку матриці D.

Складність підрахунку Евклідової метрики між початковою та факторизованою моделлю складає $O(nm)$. Такий підрахунок просто реалізується з використанням графічних адаптерів.

Розподілений алгоритм

Наступним кроком для покращення швидкодії є використання мережі ПК для проведення обчислень. Розглянемо можливі розподіленні моделі. Припустимо, що мережа складається з двох вузлів. Можливі наступні моделі розподілення обчислень:

1. *Підраховувати W та H' окремо на різних вузлах.* Таким чином обидва вузли будуть перебувати в одному з двох альтернативних режимів. Або будуть вузлом підтримки іншого вузла (передача даних на інший вузол) або будуть проводити підрахунки (обчислювати матрицю, за яку відповідають, за допомогою даних, отриманих з вузла підтримки). У такій моделі розподілення на кожній ітерації ми повинні будемо передати по мережі дві матриці такого ж розміру, як і W та H. Також вузол розрахунків буде в основному простоювати, тому що (7a) є найбільш витратним кроком з усіх чотирьох.

2. *Розділити матриці W та H' на блоки та розподілити їх між вузлами.* $H' = (H_1' | H_2')$ та $W = (W_1 | W_2)$. Робимо перший вузол відповідальним за H_1', W_1, другий за H_2', W_2. При такому підході кожен вузол виступає як у ролі вузла підтримки, так і у ролі вузла розрахунків одночасно. У такій моделі вузли повинні передавати по мережі об'єм даних рівний $1.5 \cdot (sizeof\,(W) + sizeof\,(H))$, де $sizeof\,(X)$ – об'єм пам'яті, необхідний для зберігання матриці X

3. *Розділити матриці W та H' на блоки та розподілити їх між вузлами.* $H' = (H_1 | H_2)^T$ та $W = (W_1' | W_2')^T$. Робимо перший вузол відповідальним за H_1, W_1', другий за H_2, W_2'. У цій моделі, аналогічно до попередньої, кожен вузол мережі виступає одночасно як у ролі вузла підтримки, так і у ролі вузла розрахунків. Об'єм передачі даних вузлом рівний $sizeof\,(W) + sizeof\,(H)$

У кожній з представлених моделей також є необхідність передачі одної або декількох матриць [k;k], але їх розміром можна знехтувати в порівнянні з матрицями W та H. Для підрахунку функції оцінки μ в першій та третій моделі вузлам необхідно передати дані в об'ємі $\frac{(sizeof(W) + sizeof(H)) * K}{2}$, де K – кількість вузлів в мережі, у другій моделі необхідно передати в $(K - 1)$ раз більше даних.

Варто також зазначити, що ріст мережі призведе до експоненціального росту сумарного об'єму переданих даних, хоча об'єм передачі даних одного вузла буде не більш ніж $2 \cdot (sizeof\,(W) + sizeof\,(H))$.

Для кращого розподілення розрахунків між вузлами мережі, враховуючи розрідженість початкової матриці V, необхідно виконати перестановки рядків та стовпчиків для нормалізації кількості ненульових елементів у кожному блоці.

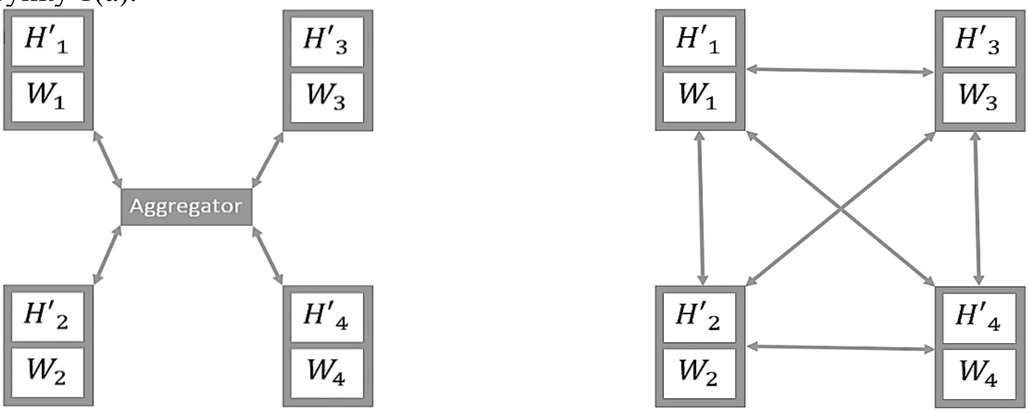
Очевидно, що третя модель найкраще підходить з точки зору мережевого обміну даними та використання GPU. Тому саме ця модель була використана в нашому дослідженні, реалізована з використанням мережі з чотирьох вузлів і описана далі.

Згідно з моделлю 3 необхідно розділити матриці $W, H`$ і Vнаступним чином: $W = \left(W'_1 \middle| W'_2 \middle| W'_3 \middle| W'_4\right)^T$, $H' = (H_1|H_2|H_3|H_4)^T$, $V = \begin{pmatrix} V_{11} & \cdots & V_{14} \\ \vdots & \ddots & \vdots \\ V_{41} & \cdots & V_{44} \end{pmatrix}$.

Алгоритм складається з трьох основних фаз: ініціалізації, ітерації, підрахунку метрики. Під час фази ініціалізації матриці $W, H`$ і V розподіляються між чотирма вузлами так, щоб i-ий вузол отримав матриці W'_i, H_i та V_{ki}, V_{ik}, де $k = \overline{1,\ldots,4}$. Дана фаза представлена на рисунку 1(а).

Фаза ітерації складається з двох аналогічних етапів, один для підрахунку H', а другий для W. Кожен етап включає 3 кроки.

На першому кроці кожний вузол підраховує матрицю $W'_i \times W'^T_i$розміру $k \times k$ та надсилає її до агрегатору. Агрегатор об'єднує всі отримані блоки в одну матрицю K_w та надсилає отриманий результат всім вузлам. Цей крок представлено на Рисунку 1(а).



(a) (b)

Рис. 1. Розподілена модель з чотирма вузлами

На другому кроці кожен вузол підраховує власне $\left(V^T_{1i} \middle| V^T_{2i} \middle| V^T_{3i} \middle| V^T_{4i}\right)^T \times W'_i$. Отримана матриця співпадає за розмірами з H'. Далі кожний вузол ділить цю матрицю згідно з початковим розбиттям матриці H та передає отримані блоки відповідним вузлам. Цей крок представлено на Рисунку 1(b).

На третьому кроці вузли підраховують матрицю $H_i \times K_w$ та виконують оновлення матриці H'. Цей крок не потребує передачі даних по мережі.

Ці три кроки необхідні для підрахування матриці H'. Після поновлення H', аналогічні кроки необхідно виконати для W. А саме необхідно підрахувати наступні матриці: $H_i \times H^T_i$, $\left(V^T_{1i} \middle| V^T_{2i} \middle| V^T_{3i} \middle| V^T_{4i}\right)^T \times H_i$ та $W'_i \times K_H$.

У фазі підрахунку метрики кожен вузол передає відповідний блок H' всім іншим блокам. Після отримання всіх блоків, кожний вузол підраховує відповідну части метрики. Ця фаза також представлена на Рисунку 1(b).

Результати аналізу

Був реалізований розподілений алгоритм з використанням GPU, який був описаний вище, та проведена факторзація розрідженої матриці оцінок части вживання слів англійської мови в різних статтях Вікіпедії. Для порівняння швидкодії була також реалізована локальна модель з використанням GPU та жорсткого диску для запису даних.

Обидві моделі працювали з однаковими вхідними даними. Локальна версія була запущена на одному з вузлів з тими ж обмеженнями пам'яті. Для проведення тестів були використані наступні апаратні засоби: Intel Core i7 CPU, NVIDIA GeForce GTX560 1Gb, 8Gb RAM, 1Gbit LAN, SATA III HD.

У таблиці 2 порівняно час та ресурси, необхідні для виконання однієї ітерації в локальній та розподіленій версії. У таблиці 3 порівняно час та ресурси, необхідні для підрахунку функції оцінки μ. Дані таблиць отриманні при $k = 300$.

Таблиця 2. Результати порівняння виконання ітерації локальної та розподіленої версії

	Local	Distributed
Прочитано	34.44Gb	6.22Gb
Записано	16.58Gb	6.22Gb
Час ітерації (розрахунки)	58s	62s
Час ітерації (I/O)	729s	287s

Таблиця 3. Результати порівняння обчислення функції оцінки μ

	Local	Distributed
Прочитано	13.66Gb	6.22Gb
Записано	0	6.22Gb
Час (розрахунки)	45865s	11371s
Час (I/O)	192s	280s

На рис.2 показано графік отриманої залежності якості результатів факторизації від k та кількості ітерацій.

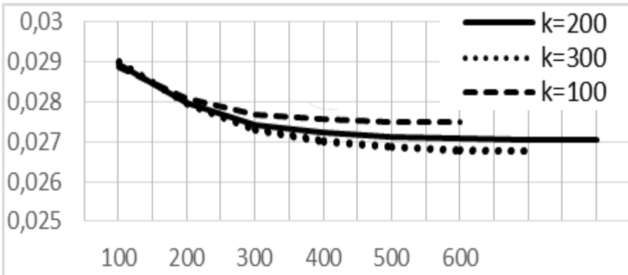


Рис 2. Графік залежності збіжності алгоритму факторизації від k та кількості ітерацій.

Застосування моделі до факторизації тензорів

Для того, щоб обчислити невід'ємні матриці-компоненти {A, B, C}, зазвичай застосовується обмежений оптимізаційний підхід для мінімізації функції оцінки, що підходить. Зазвичай мінімізують наступну функцію:

$D_F(\underline{Y}||[\![A,B,C]\!]) = ||\underline{Y} - [\![A,B,C]\!]||_F^2 + \alpha_A||A||_F^2 + \alpha_B||B||_F^2 + \alpha_C||C||_F^2$, де $\alpha_A, \alpha_B, \alpha_C$ – невід'ємні регуляційні параметри.

Існує щонайменше три різні підходи до розв'язання такої оптимізаційної задачі. Перший підхід полягає у використанні векторної форми функції оцінки: $J(X) = vec(\underline{Y} - [A,B,C]) = 0$. Для розв'язання використовується метод найменших квадратів. Цей метод для факторизації тензорів вперше використав Паатеро [11]. Якобіан такої функції може мати розмір $ITQJ \times (I + T + Q)$, а, отже, такий підхід потребує значних розрахункових витрат.

У другому підході Акар, Колда та Дунлаві запропонували штучно оптимізувати функцію оцінки, використовуючи техніку нелінійної зв'язної градієнтної оптимізації[1].

Третім, і найпоширенішим, підходом є використання техніки ALS. У цьому підході підраховується градієнт функції оцінки для кожної матриці окремо.

$\nabla_A D_F = -Y_{(1)}(C \odot B) + A\left[(C^T C) \circledast (B^T B) + \alpha_A I\right]$,

$\nabla_B D_F = -Y_{(2)}(C \odot A) + B\left[(C^T C) \circledast (A^T A) + \alpha_B I\right]$,

$\nabla_C D_F = -Y_3(B \odot A) + C\left[(B^T B) \circledast (A^T A) + \alpha_C I\right]$.

Прирівнюючи кожну компоненту градієнта до 0 та накладаючи умову невід'ємності, можна отримати ефективні та прості правила оновлення матриць:

$A \leftarrow \left[Y_{(1)}(C \odot B) + \left[(C^T C) \circledast (B^T B) + \alpha_A I\right]^{-1}\right]_+$,

$B \leftarrow \left[Y_{(2)}(C \odot A) + \left[(C^T C) \circledast (A^T A) + \alpha_B I\right]^{-1}\right]_+$,

$C \leftarrow \left[Y_{(3)}(B \odot A) + \left[(B^T B) \circledast (A^T A) + \alpha_C I\right]^{-1}\right]_+$.

Основними перевагами такого підходу є висока швидкість збіжності та можливість розподілення для задач великої розмірності.

Запропонована вище розподілена модель може бути застосована для факторизації тензорів. Правила оновлення є однаковими. Тому буде показано підрахунок на прикладі однієї матриці:

$A \leftarrow \left[Y_{(1)}(C \odot B) + \left[(C^T C) \circledast (B^T B) + \alpha_A I\right]^{-1}\right]_+$

Для мережі з двох вузлів необхідно розділити матриці W та H' на блоки таким чином, що $C' = (C_1 | C_2)^T$ та $B = (B_1' | B_2')^T$, та розподілити їх між вузлами так, щоб перший вузол був відповідальним за C_1, B_1', другий за C_2, B_2'.

При такому розподіленні на першому кроці кожний вузол підраховує матрицю $C_i^T \times C_i$ розміру $k \times k$ та надсилає її до агрегатору. Агрегатор об'єднує всі отримані блоки в одну матрицю K_C та надсилає відповідні частини до вузлів. Далі аналогічним чином виконуються розрахунки для $(B^T B)$. На наступному кроці вузлам необхідно підрахувати $S_i = \left[(C^T C)_i \circledast (B^T B)_i + \alpha_A I\right]$ та передати відповідні матриці на агрегат, на якому підрахується S^{-1}. Після чого вузлам необхідно підрахувати відповідні частини результуючої матриці у вигляді $(C \odot B)_i + S^{-1}_i$.

Таким чином, можливе використання запропонованої розподіленої моделі до факторизації тензорів.

Оптимізація факторизації матриці зв'язності

Так як розглядувана розріджена матриця є матрицею зв'язності Слова×Статті, отже можливі перестановки як рядків, так і стовпців початкової матриці.

Розглядувана розріджена матриця має 156,236,043 ненульових елементів з 10,907,060,852,120 що складає лише 0,014%. Така розрідженість дозволяє привести матрицю до блочно-діагональної форми за допомогою перестановок рядків та стовпчиків. У випадку, коли слово може належати декільком блокам, рядок може бути розділений.

Розміри блоків найкраще обирати такими, що можуть бути повністю завантажені в пам'ять. У результаті буде отримано N блоків.

Переваги такого підходу:

1. Не потрібно мережевих операцій на кожній ітерації. Необхідне лише початкове розподілення блоків матриці між вузлами.

2. Швидша збіжність. На рис. 3 та рис. 4 показано графіки залежності часу та кількості ітерацій відповідно, необхідні для факторизації квадратних матриць в один потік без використання GPU.

3. Швидший підрахунок функції оцінки μ. Кількість елементів для підрахунку функції оцінки менша в N разів, кількість необхідних перевірок менша, так як зменшується загальна кількість ітерацій.

4. Зменшення K, а, отже, і зменшення розмірів результатів.

У результаті для кожного слова буде отримано один або декілька рядків та ідентифікатор блоку, в якому це слово знаходиться. При відновленні, якщо слово та стаття знаходяться в різних блоках – результатом буде 0, інакше результатом буде добуток відповідних рядка та стовпчика.

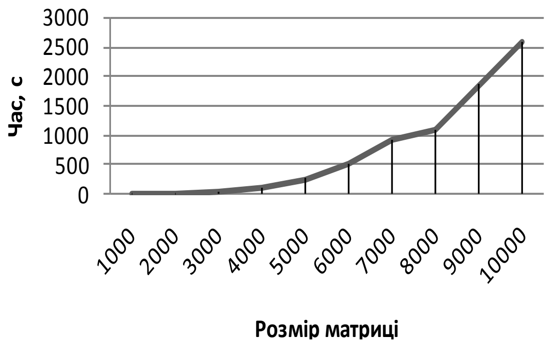


Рис. 3. Графік залежності часу ітерацій від розміру матриці

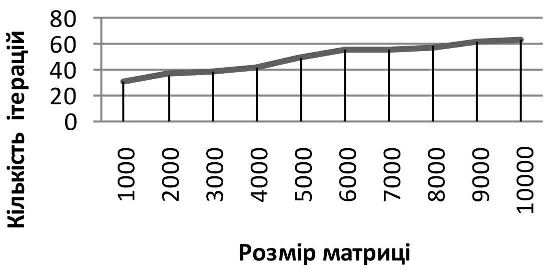


Рис. 4. Графік залежності кількості ітерацій від розміру матриці

Література

1. Xu, W., Liu, X., & Gong, Y. (2003). Document-clustering based on XXXn-negative matrix factorization. In Proceedings of SIGIR'03, July 28–August 1, Toronto, CA, pp. 267–273.
2. Farial Shahnaz, Michael W. Berry, V. Paul Pauca, Robert J. Plemmons Document clustering using nonnegative matrix factorization// Information Processing and Management: Volume 42 Issue 2, March 2006 P.P. 373 – 386
3. Anatoly Anisimov, Oleksandr Marchenko, Andrey Nikonenko, Elena Porkhun, Volodymyr Taranukha: Ukrainian WordNet: Creation and Filling. FQAS 2013: 649-660
4. Tim Van de Cruys. 2010. Anon-negative tensor factor-ization model for selectional preference induction. Natural Language Engineering, 16(4):417–437.
5. Tim Van de Cruys, Laura Rimell, Thierry Poibeau, and Anna Korhonen. 2012. Multi-way Tensor Factorization for Unsupervised Lexical Acquisition. In International Conference on Computational Lin-guistics (COLING), Mumbai, India, 08/12/2012-15/12/2012, pages 2703–2720. The COLING 2012 Organizing Committee.
6. Scott Deerwester, Susan T. Dumais, GeorgeW. Furnas, Thomas K. Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE, 41(6):391–407.
7. Brett W. Bader, Tamara G. Kolda, et al. 2012. Matlab tensor toolbox version 2.5. Available online, January.
8. Khushboo Kanjani. 2007. Parallel non negative matrix factorization for document clustering. May.
9. Volodymyr Kysenko, Karl Rupp, Oleksandr Marchenko, Siegfried Selberherr, and Anatoly Anisimov. 2012. Gpu-accelerated non-negative matrix factorization for text mining. volume 7337 of Lecture Notes in Computer Science, pages 158–163. Springer.
10. T.K. Landauer, P.W. Foltz, and D. Laham. 1998. An introduction to latent semantic analysis. Discourse processes, 25:259–284.

11. Daniel D. Lee and H. Sebastian Seung. 2000. Algorithms for non-negative matrix factorization. In In NIPS, pages 556–562. MIT Press.
12. Chao Liu, Hung-chih Yang, Jinliang Fan, Li-Wei He, and Yi-Min Wang. 2010. Distributed nonnegative matrix factorization for web-scale dyadic data anal-ysis on mapreduce. In Proceedings of the 19th In-ternational Conference on World Wide Web, WWW'10, pages 681–690, New York, NY, USA. ACM.
13. Rada Mihalcea, Courtney Corley, and Carlo Strappa-rava. 2006. Corpus-based and knowledge-based measures of text semantic similarity. In IN AAAI06, pages 775–780, Menlo Park, CA; Cambridge, MA; London. AAAI Press;.
14. nVidia, 2013a. CUBLAS Library User Guide. nVidia, v5.0 edition, October.
15. nVidia, 2013b. CUDA CUSPARSE Library. nVidia, August.
16. Farial Shahnaz, Michael W. Berry, V. Paul Pauca, and Robert J. Plemmons. 2006. Document clustering using nonnegative matrix factorization. Inf. Process. Manage., 42(2):373–386, March.
17. Wei Xu, Xin Liu, and Yihong Gong. 2003. Document clustering based on non-negative matrix factorization. In Proceedings of the 26th Annual Inter-national ACM SIGIR Conference on Research and Development in Informaion Retrieval, SIGIR '03, pages 267–273, New York, NY, USA. ACM.

RESUME

E.M. Nasirov

Model for parallel non-negative sparse extra-large matrix factorization

This paper proposes parallel methods of non-negative large sparse matrix factorization - very popular technique in computational linguistics. The problem of non-negative factorization for a sparse large matrix emerged in the development of a measure of semantic similarity between words with Latent Semantic Analysis usage. Previously developed parallel models for Non-Negative Matrix Factorization do not meet the requirements of the matrix dimensions or requires excessively large computational resources.

It is difficult to execute NMF algorithm on a single machine, without dumping data to the hard drive due to excessive memory requirements. Two variants of the algorithm implementation are described: local with intensive hard drive and GPU usage and distributed with intensive network and GPU usage. Basic iterations rules were divided to steps in order that requires a minimal number of calculations in a manner that reduces amount of excessive memory copying.

Three distribution models were compared. Memory usage and data transmitting necessity of factorization algorithm was analyzed and optimized. The described effective GPU-based and distributed algorithms were implemented, tested on the grid of four nodes and compared by means of large sparse matrices processing.

GPU-based and distributed algorithms were combined, with special attention to memory usage, which allows larger input matrices to be factorized. The experiments showed the constructed model is effective. It can be used to perform the tasks of industrial scale to factorize sparse matrices of large dimension with an acceptable time using available computing resources. This paper describes modification of proposed distributed model of matrix mactorization to be used for fast and effective non-negative factorization of large sparse tensors.

Paper proposes blocks-diagonal approach for factorization of sparse extra-large matrices which can be transformed to blocks-diagonal form. This approach can speed up factorization, is easy paralleling, needs less network operations and memory resources for iterations and results storage.

Е.М. Насіров

Модель паралелізації невід'ємної факторизації розріджених матриць надвеликої розмірності

У цій статті запропоновано паралельні методи невід’ємної факторизації надвеликих розріджених матриць – популярний метод в комп'ютерній лінгвістиці. Проблема невід’ємної факторизації розріджених матриць постала в процесі розробки системи визначення міри семантичної близькості-зв’язності за технологією Латентного Семантичного Аналізу. Існуючі паралельні моделі для невід’ємної факторизації матриць не задовольняють потреби розмірності матриці або вимагають занадто великих обчислювальних ресурсів.

Описано два варіанти реалізації алгоритму: локальний з використанням жорсткого диска та GPU і розподілений з використанням мережі та використання GPU. Основні правила ітерації були розділені на кроки з метою, яка вимагає мінімальної кількості обчислень таким чином, що знижує кількість надлишкового копіювання пам'яті.

Були зіставлені три моделі розподілу. Використання пам'яті і об’єми передачі даних, необхідні для роботи алгоритму факторизації були проаналізовані та оптимізовані. Описані локальна модель з використанням GPU та розподілена модель були реалізовані, випробувані та порівняні.

Локальний та розподілений алгоритми були об'єднані, з особливою увагою до використання пам'яті, що дозволяє факторизувати вхідні матриці надвеликої розмірності. Експерименти показали, що побудована модель є ефективною. Вона може бути використана для виконання завдань промислового масштабу по факторізації розріджених матриць надвеликої розмірності за прийнятний час, використовуючи доступні обчислювальні ресурси.

Ця стаття описує модифікацію запропонованої розподіленої моделі для швидкої та ефективної факторизації розріджених тензорів великої розмірності.

У статті запропоновано блочно-діагональний підхід до факторизації розріджених матриць, які можуть бути приведені до блочно-діагональної форми. Цей підхід може прискорити факторизацію, потребує менше мережевих операцій та пам'яті для ітерацій і зберігання результатів.

Надійшла до редакції 30.08.2015

